Remarks

Reconsideration of the application is respectfully requested in view of the foregoing amendments and following remarks. Claims 1, 3-10, 12-15, 17-24, and 26-32 are pending in the application. No claims have been allowed. Claims 1, 6, 12, 14, and 24 are independent. Claims 12 and 24 have been amended. Claim 25 has been canceled without disclaimer and without prejudice to pursuing in a continuing application.

Cited Art

The Office action ("Action") applies the following cited art: U.S. Patent No. 6,981,249 to Knoblock et al. ("Knoblock") and U.S. Patent No. 7,117,488 to Franz et al. ("Franz").

§ 102 Rejection

The Action rejects claims 1, 3-8, 12-14, and 17-31 under 35 U.S.C. § 102(e) as being anticipated by Knoblock. Applicant respectfully submits that the claims are allowable over the cited art. To establish a prima facie case of anticipation, the cited art must show each and every element as set forth in a claim. MPEP § 2131.01.

Claims 1 and 14

Claim 1 recites (emphasis added):

> A method of *type-checking* a code segment written in a programming language comprising:
> translating the code segment from the programming language to one or more representations of an intermediate language; and
> *type-checking the one or more representations based on a rule set, wherein the rule set comprises rules for type-checking a type designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known.*

Claim 14 recites (emphasis added):

> A system for *type-checking* an intermediate representation of source code in a compiler comprising:
> one or more types associated with elements of the intermediate representation, wherein at least one of the types, designated as an unknown type, indicates an element can be of any type;
> *one or more rule sets comprising rules associated with the type,*

designated as the unknown type, indicating an element can be of any type; and
a type-checker for applying the one or more rule sets to the elements of the intermediate representation.

Knoblock's description of reconstructing types does not teach or suggest "A method of type-checking" comprising "type-checking the one or more representations based on a rule set, wherein the rule set comprises rules for type-checking a type designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known," as recited by claim 1.

Knoblock is directed to type reconstruction, not type checking. As Knoblock describes, when a source program is translated (e.g., compiled) to produce a bytecode program, some types that were present in the source program are lost. Knoblock, col. 5, lines 59-66. Knoblock states that before type checking can begin, a program must have types. Knoblock, col. 5, lines 58-59. Therefore, Knoblock provides a solution for reconstructing types that were lost during translation of a source program to a bytecode program. Knoblock, col. 5, line 58 to col. 6, line 13. The solution described by Knoblock for reconstructing types involves a constraint collection technique. Specifically, Knoblock states, "the embodiments of the present invention provide a constraint collection technique to learn from the remaining portions of the method 320 to solve for the type of the local variable x." Knoblock, col. 7, lines 21-31. Knoblock describes one method for reconstructing types involving labeling variables as unknown, collecting constraints between known types and unknown types, and solving for the unknown types using the constraints. Knoblock, col. 8, lines 4-55.

Knoblock's does not teach or suggest "A method of *type-checking*" comprising "*type-checking the one or more representations based on a rule set, wherein the rule set comprises rules for type-checking a type designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known*" as recited by claim 1. Instead, as described above, Knoblock reconstructs type information lost during translation. Knoblock states that type reconstruction must be performed before type checking can begin. Rather than reconstructing type information before type checking, claim 1 recites a method for type checking using rules for checking unknown types.

Furthermore, Knoblock describes the advantages of type checking to ensure that strong typing is enforced. Knoblock, col. 5, lines 45-57. Knoblock further states, "the debugger is also adapted for type checking the intermediate program using the reconstructed types so as to enhance the identification of faults." Knoblock, col. 6, lines 25-29. Therefore, even if type checking were to be performed after using Knoblock's solution for reconstructing type information, the type checking would use reconstructed types (i.e., known types). Instead of type checking using reconstructed types, claim 1 recites, "rules for type-checking a type designated as an unknown type."

For at least these reasons, Knoblock does not each or suggest the language of claim 1. Therefore, claim 1 should be in condition for allowance.

In addition, for at least the reasons discussed above with regard to claim 1, Knoblock does not teach or suggest "A system for type-checking" comprising "one or more rule sets comprising rules associated with the type, designated as the unknown type, indicating an element can be of any type; and a type-checker for applying the one or more rule sets to the elements of the intermediate representation," as recited by claim 14. Therefore, claim 14 should be in condition for allowance.

Claim 6

Claim 6 recites (emphasis added):

> A method of *selectively retaining type information* during compilation in a code segment written in a programming language, the method comprising:
> translating the code segment from the programming language to one or more representations of an intermediate language;
> for each representation, *determining whether to retain type information for one or more elements of the representation*; and
> *based on the determination, associating one or more elements of the representation with a type, designated as an unknown type, indicating the element can be of any type.*

Knoblock's description of reconstructing types does not teach or suggest "A method of *selectively retaining type information*" comprising "*determining whether to retain type information for one or more elements of the representation; and based on the determination, associating one or more elements of the representation with a type, designated as an unknown type, indicating the element can be of any type*," as recited by claim 6.

The Examiner cites to Knoblock at col. 8, lines 8-55, and Fig. 4, as describing the above-cited language of claim 6. However, as described above with reference to claim 1, this section of

Knoblock describes reconstructing types by solving for unknown types using constraints. Knoblock does not teach or suggest "determining whether to retain type information for one or more elements of the representation; and based on the determination, associating one or more elements of the representation with a type, designated as an unknown type, indicating the element can be of any type." In fact, Knoblock does just the opposite; Knoblock solves for unknown types.

For at least these reasons, Knoblock does not each or suggest the language of claim 6. Therefore, claim 6 should be in condition for allowance.

Claims 12 and 24

Claim 12, as amended, recites (emphasis added):

> A method of translating types associated with a plurality of programming languages to types of an intermediate language, the method comprising:
>
> replacing the types associated with the plurality of programming languages with the types of the intermediate language, wherein the types of the intermediate language comprise general categories of the types associated with the plurality of programming languages and a type designated as an unknown type, *wherein the type designated as the unknown type has size information associated with it, wherein the size information comprises size information of a machine representation of the type designated as the unknown type.*

Claim 24, as amended, recites (emphasis added):

> A method of representing types in an intermediate language comprising:
>
> defining a plurality of types to be associated with elements of the intermediate language, wherein one of the plurality of types indicates that an element of the intermediate language is associated with a type designated as an unknown type;
>
> *wherein the type indicating that an element of the intermediate language is associated with the type designated as the unknown type has a size associated with it, wherein the size represents size of a machine representation of the type designated as the unknown type.*

Claim 24 has been amended with language from dependent claim 25 and from claim 32. In addition, see the Application at page 9, lines 11-15. Claim 12 has been amended with similar language.

As stated by the Examiner, Knoblock does not describe a size for a type designated as an unknown type (or a size of a machine representation of the type). Action, pages 7 and 8, with reference to claims 9, 10, 15, and 32. Instead, The Examiner argues that Franz teaches size information, citing Franz description of array size at col. 11, line 63 to col. 12, line 11. Applicants respectfully disagree. Franz describes size with relation to array indexing (bounds), so that array indexes are safe. Franz, col. 11, line 63 to col. 12, line 11. Franz describes the need for enforcing array index checking at col. 1, line 60 to col. 2, line 3:

> Assume an array is declared as having ten elements. A malicious code provider might wish to access array element eleven, thereby circumventing the type rules and gaining access to whatever variable happened to be located at the memory location corresponding to element eleven of the array--even if that variable is marked as being private or protected. Many exploits of security holes use this route, using a breach of type safety to modify variables that they normally would not have access to. Type safe code prevents this by disallowing the referencing of array elements beyond those defined to be in the array.

Franz does not describe "*wherein the type designated as the unknown type has size information associated with it, wherein the size information comprises size information of a machine representation of the type designated as the unknown type*," as recited by claim 12, or "*wherein the type indicating that an element of the intermediate language is associated with the type designated as the unknown type has a size associated with it, wherein the size represents size of a machine representation of the type designated as the unknown type*," as recited by claim 24.

For at least these reasons, Knoblock separately or in combination with Franz, does not teach or suggest the above-cited language of claims 12 and 24, respectively. Therefore, claims 12 and 24 should be in condition for allowance.

Claims 3-5, 7, 8, 13, 17-23, 26-31

Claims 3-5, 29, and 30 depend on claim 1. Thus, for at least the reasons set forth above with regard to claim 1, claims 3-5, 29, and 30 should be in condition for allowance.

Claims 7, 8, and 31 ultimately depend on claim 6. Thus, for at least the reasons set forth above with regard to claim 6, claims 7, 8, and 31 should be in condition for allowance.

Claim 13 depends on claim 12. Thus, for at least the reasons set forth above with regard to claim 12, claim 13 should be in condition for allowance.

Claims 17-23 ultimately depend on claim 14. Thus, for at least the reasons set forth above with regard to claim 14, claims 17-23 should be in condition for allowance.

Claims 26-28 depend on claim 24. Thus, for at least the reasons set forth above with regard to claim 24, claims 26-28 should be in condition for allowance.

§ 103 Rejection

The Action rejects claims 9, 10, 15, and 32 under 5 U.S.C. § 103(a) as unpatentable over Knoblock in view of Franz. Applicant respectfully submits that the claim is allowable over the cited art.

Claims 9 and 10 ultimately depend from claim 6, claim 15 depends from claim 14, and claim 32 depends from claim 1. Therefore, for at least the reasons stated above with regard to claims 1, 6, and 14, Knoblock does not teach or suggest the language of claims 9, 10, 15, and 32 (as dependent on their respective independent claims) cited above. Furthermore, as understood by Applicants, Franz does not add sufficient disclosure to teach or suggest this language. Thus, claims 9, 10, 15, and 32 should be in condition for allowance.

Request for Interview

If any issues remain, the Examiner is formally requested to contact the undersigned attorney prior to issuance of the next Office action in order to arrange a telephonic interview. It is believed that a brief discussion of the merits of the present application may expedite prosecution. Applicants submit the foregoing formal Amendment so that the Examiner may fully evaluate Applicants' position, thereby enabling the interview to be more focused.

This request is being submitted under MPEP § 713.01, which indicates that an interview may be arranged in advance by a written request.

Conclusion

The claims should be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 595-5300
Facsimile: (503) 595-5301

By /Cory A. Jones/
Cory A. Jones
Registration No. 55,307